KATHOLIEKE
UNIVERSITEIT
LEUVEN

DEPARTEMENT TOEGEPASTE ECONOMISCHE WETENSCHAPPEN

RESEARCH REPORT 0015

MEASURES FOR OBJECT-EVENT INTERACTIONS

by
G. POELS
G. DEDENE

D/2000/2376/15

Measures for Object-Event Interactions

Geert Poels, Guido Dedene

Management Information Systems Group
Departement of Applied Economic Sciences
Katholieke Universiteit Leuven
Naamsestraat 69, B-3000 Leuven, Belgium
{geert.poels, guido.dedene}@econ.kuleuven.ac.be

Abstract. A suite of measures is presented that addresses two problem areas within contemporary object-oriented software measurement theory and practice, i.e. the lack of OOA measures and the lack of measures for behavioural aspects of software. Our suite of measures is based on a formally defined model of object-event interaction, called the object-event association matrix. Generally, the objects in a domain or system are affected by the occurrence of real-world or information-system events. A framework for measurement is presented that expresses and measures attributes (features, properties, characteristics) of objects, related to the data, function and dynamic behaviour dimensions of software, in terms of object-event interactions. This framework allows for early measurement because it applies to OOA, CBSE, and domain analysis methods that support the notion of events, or that support similar dynamic concepts (use cases, scenarios, actions) that can be transformed into events.

1. Introduction

Recently, two workshops on object-oriented software measurement gave a mixed picture of the maturity of the field [7, 16]. The good news is that now, at the turn of the millennium, we have a massive amount of measures for object-oriented software artifacts to choose from. In his recent book, Horst Zuse has listed more than 200 of such measures [37]. The other good news is that a substantial subset of these measures is supported by measurement tools and has effectively been used in empirical software engineering research to investigate various aspects of software quality. The bad news is that there are still areas that remain largely untouched by object-oriented software measurement research.

In this paper we focus upon two of these problem areas. First, there are almost no measures for the dynamic aspects of object-oriented software. Measure suites like MOOSE [17], MOOD [13], QMOOD [2] and C-FOOD [10] focus on the data and function dimensions of software. They include measures like CBO (Coupling Between Objects), DIT (Depth of Inheritance Tree) and LCOM (Lack of COhesion in Methods), which assume models of object-oriented software that detail the associations between classes, the inheritance tree or lattice, the message passing structure, and the attributes referenced within the class methods, but that abstract from behavioural dynamics aspects as modeled, for instance, in statechart diagrams and activity diagrams. Second, although industry begs for measurement instruments that can be applied in the early phases of the development process (mainly for early quality control and project budgeting decisions), nearly all published object-oriented software measures can only be used after (high-level) system design. Some exceptions known to us are Graham's task points [22] and the QOOD measure suite for object-oriented analysis [1].

We believe these two problem areas to be somewhat related. Currently, there is a lack of measurement support for modern UML-compliant approaches towards domain analysis, OOAD and CBSE, like Catalysis [20] and the Rational Unified Process [30]. An important feature of these approaches is that during domain and systems analysis a strong emphasis is put on dynamic aspects. Concepts like uses cases, scenarios, actions and events are not only used to discover the relevant domain objects, but also to model the interactions between these objects (e.g. in a collaboration diagram), and thus to shape the overall architecture of the system. Also, in the previous generation of OOAD methods (e.g. Fusion [18], Syntropy [19], OO-SSADM [31]), events are used as building blocks for modeling a domain or a system.

In this paper, we present a conceptual measurement framework that is based on a model of object-event interaction. Generally, the objects in a domain or system are affected by the occurrence of real-world or information-system events. As an example, consider an ORDER object that can be placed, changed, delivered, invoiced, paid, etc. (real-world events), or entered into the system, queried, modified, moved to a data warehouse, destroyed, etc. (information-system events). In our framework, a fairly simple model of such event participations is proposed. This model, called the object-event association matrix, is sufficient to express and measure a whole range of software attributes (e.g. size, functionality, complexity, coupling, etc.) and the actual use of OO mechanisms (e.g. inheritance, overriding, static and late binding, polymorphism, etc.), in terms of object-event interactions. An important feature of the framework is that the object-event association matrix is derived during object-oriented analysis. The framework thus provides a means to define measures for assessing the attributes of object-oriented analysis artifacts (e.g. object types), as well as for estimating the attribute values of artifacts that are developed from the OOA artifacts (e.g. object classes).

Another feature of the framework is that it allows measuring dynamic aspects such as object life cycle complexity and synchronisation-based coupling.

In section 2 we present key techniques and artifacts related to object-oriented analysis. In particular we emphasise the role of the object-event association matrix for this type of modeling. For the sake of clarity and simplicity, we restrict the scope to conceptual modeling, i.e. the concepts presented in section 2 can be used to model, structure and analyse a (part of a) domain, irrespective of the information system that must be built. Object-oriented analysis in general also aims at modeling and analysing the specific information system requirements (user interface, data storage, workflow aspects, quality requirements, etc.). The conceptual measurement framework and the object-event association matrix itself can easily be extended, without loss of validity or genericity, to include information-system events and objects other than domain objects. It must be noted that, due to the very nature of our framework, our view on conceptual modeling goes further than pure data modeling. Behavioural modeling is an essential part of the process.

In section 3 the conceptual measurement framework itself is presented, along with a compact measure suite that includes size, coupling, inheritance, specialisation, propagation and polymorphism measures. This measure suite is not exhaustive. We had to limit the number of measures due to space limitations. A few of the measures not included here, but especially relevant to measure dynamic aspects of software (i.e. object class life cycle complexity), have been published elsewhere [27].

The paper ends with conclusions and a few suggestions for further research (section 4).

2. Event-driven OO conceptual modeling: definitions and notations

Definitions and notations are presented for a number of concepts relevant to object-oriented conceptual modeling. We sketch a modeling method that is sufficiently abstract and generic to capture the main aspects of this type of modeling, and pay special attention to the dynamic aspects of the model. Our 'archetype' conceptual modeling method has been fully described in [35]. This method has been called 'event-driven' in [33] because of the importance it attaches to events.

2.1. Events

In conceptual modeling, the events that are modeled are real-world events, sometimes also referred to as business events. Real-world events are characterised by the following properties:

- A real-world event corresponds to something that happens in the real world. This 'real world' is the universe of discourse, i.e. the domain or relevant part of the domain that must be modelled. This property allows real-world events to be distinguished from information-system events, which depend upon the presence of an information system.
- A real-world event has no duration, i.e. it occurs or is recognised at one point in time. This property allows events to be distinguished from actions, which do have a duration. The beginning and ending of an action qualify as events.
- The real-world events that are identified during conceptual modeling are not further decomposable. They are defined at the lowest level of granularity and cannot be meaningfully split into other, more fine-grained, events. This property allows events to be distinguished from use cases and scenarios.

It is common to model events as event types, rather than referring to specific event occurrences. Since real-world events are the focal point in event-driven conceptual modeling, a notation is needed to designate the set of event types that is relevant for a particular universe of discourse. A capital A is used to denote the universe of event types associated with some universe of discourse. All event types relevant to the universe of discourse are elements of A.

An example is presented of a simplified loan circulation process in the context of a library. Assume that the scope of the **LIBRARY** conceptual model is initially delimited such that the universe of event types is
A = {*start_membership, end_membership, acquire, catalogue, borrow, renew, return, sell, reserve, cancel, fetch, lose*}

2.2. Objects

A conceptual model also identifies the entities (persons, things, etc.) in the universe of discourse that participate in real-world events. Such entities are said to be 'relevant to' the universe of event types A. In object-oriented conceptual modeling these entities are represented as objects. Objects are characterised as follows:

- Each object in the conceptual model corresponds to a real-world concept.
- Objects are described by a number of properties. The properties of an object are specified in an object type (e.g. a UML classifier with an <<object type>> stereotype [6]).
- Objects exist for a certain period of time.
- An object always participates in at least two real-world events: a creating event and an ending event. The participation in the ending event does not imply that the object is physically destroyed. It means that the object can no longer participate in real-world events.

Objects have a state and a set of operations. Although the specific form of communication (e.g. messaging, broadcasting, etc.) is not relevant for conceptual modeling, we assume that for each type of event that an object participates in, there is an operation specified in the object type. The state of an object is represented by its values for the attributes that have been specified in the object type. These attributes must be seen as abstract attributes, i.e. they must not necessarily be stored attributes in the class definition of the object (cf. the principle of uniform access [25]). The effect of an event participation is modeled by specifying how the operation that is triggered by the event, affects the state of the participating object. The set of operations / triggering event types for an object type is called its *alphabet*. It is a subset of the universe of event types.

Event participations are modeled using an object-event association matrix, similar in concept to a Create-Read-Update-Delete (CRUD) matrix [24]. The *type of involvement* of an event participation is create (C), modify (M), or end (E). A modifying event type for an object type does not create object instances of the type, nor does it end their lives. A modifying event may however change the state of an object. Table 1 contains the object-event association matrix for **LIBRARY**. Apart from a type of involvement indication, we also indicate the *type of provenance* of an event participation. An operation / event type in the alphabet of an object type is either acquired through propagation (A) (cf. infra), inherited (I), or specialised, i.e. inherited in a specialised version (S) (cf. infra). The class of 'own' event types (O) completes the partitioning.

Table 1. Object-event association matrix for LIBRARY

	ITEM	VOLUME	COPY	RESERVATION	MEMBER	LOAN	NOT_RENEWABLE_LOAN	RENEWABLE_LOAN
acquire	O/C							
acquire_volume		S/C						
acquire_copy			S/C					
catalogue	O/M	I/M	I/M					
sell	O/E							
sell_volume		S/E						
sell_copy			S/E					
reserve			A/M	O/C	A/M			
cancel			A/M	O/E	A/M			
fetch			A/M	O/E	A/M			O/C
start_membership					O/C			
end_membership					O/E			
borrow					A/M	O/C		
create_not_renewable_loan		A/M			A/M		S/C	
create_renewable_loan			A/M		A/M			S/C
return		A/M	A/M		A/M	O/E	I/E	I/E
lose					A/M	O/E		
lose_volume		A/E			A/M		S/E	
lose_copy			A/E		A/M			S/E
renew			A/M		A/M			O/M

To formally define the measures in the next section, the notion of object-event association matrix is formalised [35]. The set of object types relevant to the universe of event types A is denoted by a capital T.

Let A be the universe of event types and T be the set of object types.

The object-event association matrix is a map $\tau: A \times T \rightarrow \{O, A, S, I\} \times \{C, M, E\} \cup \{(' ',' ')\}$

When $\tau(e,P) = (R,J)$ with $R \in \{O, A, S, I, ' '\}$ and $J \in \{C, M, E, ' '\}$, we write that $\tau(e,P) = R/J$.

We define the partial maps τ_P and τ_I that return the type of provenance and the type of involvement as

$\tau_P: A \times T \rightarrow \{O, A, S, I, ' '\}$

$\tau_I: A \times T \rightarrow \{C, M, E, ' '\}$

The type of involvement indications in the object-event association matrix help to derive the dynamic behaviour of objects. They specify a default life cycle. First, we have a choice between the creating event types to create an object instance. Next, the state of the object may be modified zero, one or more times, using any type of modifying event. Finally, the life of the object is ended. If sequence, selection and iteration are denoted using the ".", "+", and "*" symbols respectively, then a default life cycle for a loan object is modeled by (*borrow* + *fetch*) . *renew** . (*return* + *lose*). Additional life cycle constraints might be required, e.g. in the context of roles. We might for instance require that a book can only be borrowed if it has been catalogued first. Life cycle models can be specified in a separate model (e.g. state chart, sequence diagram, etc.). Life cycle constraints can also be specified as preconditions of operations.

2.3. Associations

Apart from dynamic aspects, conceptual modeling is also concerned with modeling the static structure of the universe of discourse. This means that associations between object types, with their optionalities and cardinalities, are identified. In event-driven conceptual modeling, the effect of associations on event participation is explicitly modeled.

One type of association is specialisation. One object type can specialise another object type. A subtype inherits the alphabet of its supertype.[1] It may also extend this alphabet, by participating in additional types of event. We also allow for event type specialisation, i.e. objects of a subtype may participate in events of a type that specialises an event type in the alphabet of the supertype (cf. Table 1).[2]

For the other types of association (e.g. aggregation, composition, etc.) we follow a modeling strategy that factors all associations into binary existence dependency associations [34]. Such associations put special restrictions on optionalities and cardinalities.[3] Moreover, they allow a formal definition of 'propagation' of operations. It is required that an object (hereafter called the master object) participates in all events in which its existence dependent objects participate. All these event participations are propagated from the existence dependent object to the master object (cf. Table 1). The operations in the existence dependent object type are also propagated into the master object type.[4]

Figure 1 shows the structural model of **LIBRARY**. Note that the object type ITEM has two subtypes: VOLUME and COPY. If we assume that volumes can be borrowed, but their loans cannot be renewed, then the object type LOAN must also be specialised.

3. An event-centric measurement framework and measure suite

In our view, conceptual modeling methods that are both object-oriented and event-driven, do not prefer the object concept above the event concept, or vice versa. Both concepts play a crucial role in the modeling process, and they can both be seen as elementary building blocks for conceptual models. In general, object-oriented analysis, CBSE, and domain engineering methods use dynamic concepts (e.g., events, actions, use cases, scenarios, etc.) when modeling the universe of discourse or the AS-IS and TO-BE information systems. The models themselves are mostly expressed in terms of objects. This might be the consequence of the principle of 'seamless development' [32], allowing the type definition of an object to be smoothly refined into a class definition. However, there are implementation schemes in which separately modeled dynamic concepts are also implemented as classes. We might for instance refine event type specifications into event classes. Such a schema makes sense if we wish to check all rules and constraints that can be attributed to an individual real-world event, before passing the event through to the object model (i.e. filtering).

---

[1] By convention, the inherited operations are explicitly specified in the subtype. This does not mean that they must be implemented in the class definition of the subtype. Conceptual modeling is not concerned with issues regarding implementation inheritance.

[2] The operations of the supertype that are specialised are not included in the subtype. Note that some object-oriented analysis methods (e.g. Catalysis [20]) do not support event type / operation specialisation.

[3] The association is mandatory and has a cardinality of one for the existence dependent object type. Moreover, an existence dependent object is during its life always associated to the same object.

[4] This does not mean that all propagated operations must also be implemented in the class definition of the master object type.

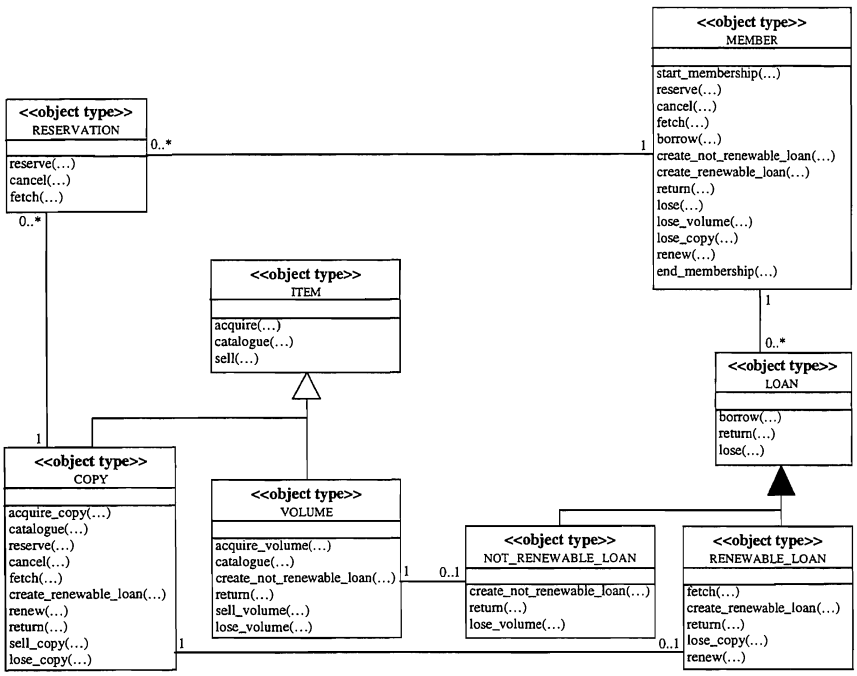

Figure 1. Structural model for LIBRARY

Nevertheless, as noted in the introduction, object-oriented software measures do generally not capture the interaction between objects and events. They do not assume models that see objects in terms of their event participations. Moreover, there are no measures for events. Therefore, we introduce here a conceptual measurement framework in which both objects and events are measurement objects, and in which the attributes measured pertain to the interaction between objects and events.

3.1. A measurement framework based on object-event interaction

Figure 2 shows part of a meta-model for the object-oriented, event-driven conceptual modeling method sketched in the previous section. The meta-model is a data model that stresses only those aspects that are within the scope of this paper.[5] In other words, the focus is on dynamic aspects related to object-event interaction.

Measurements are taken at the type level and the schema level. Measurement objects at the type level are event types and object types. In the next sub-section we present measures for characteristics of event types and object types that are a function of event participations, and their associated types of involvement (Create, Modify, End) and provenance (Own, Acquired, Specialised, Inherited). Object types are then measured in function of the types of event their object instances can participate in, and event types are measured in function of the types of object that are involved in their event occurrences. It must be noted that measures for object types could also be formulated in terms of their

[5] For the sake of clarity, we did not factor all associations into existence dependencies.

operations, given the assumption regarding the correspondence between operations and their triggering event types made in the previous section.

The measurement object at the schema level is the conceptual schema. It is an aggregate of object types and event types that are relevant for a universe of discourse (e.g., domain). The conceptual schema also contains the information regarding the interaction between object types and event types in the form of an object-event association matrix composed of event participations (from the object point of view) / object involvements (from the event point of view). This information allows direct measurement of the characteristics of conceptual schemes. Alternatively, the measurements of the more fine-grained model components (event types, object types) can be aggregated.

For either level of measurement, the object-event association matrix is the central piece of the measurement framework. Measures that capture aspects of object-event interaction are formulated by querying the object-event association matrix.

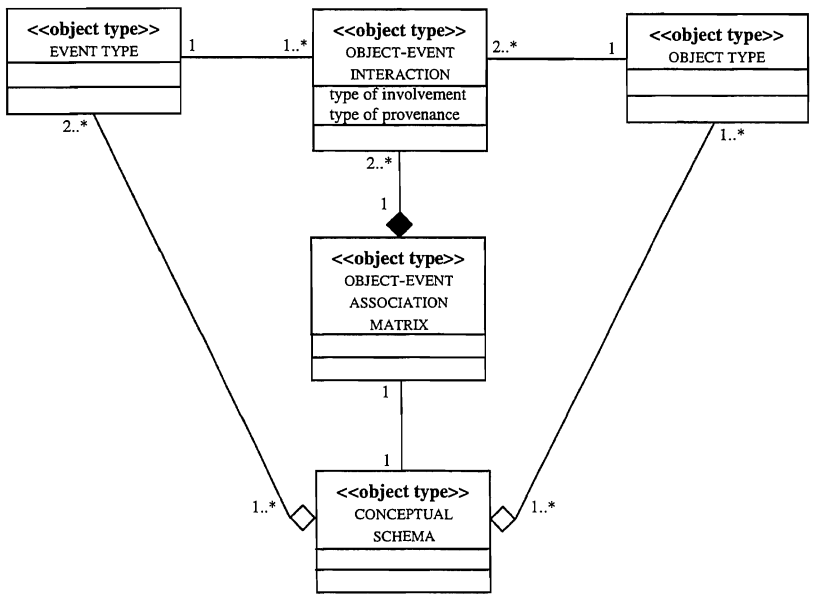

Figure 2. Part of a meta-model for event-driven OO conceptual modeling

3.2. A measure suite based on object-event interaction

The measures are grouped by measurement concept, i.e. by families of similar characteristics. Due to space limitations, we do not present elaborate definitions of these measurement concepts here.[6]

For the measure definitions, assume a universally qualified conceptual schema S with universe of event types $\boldsymbol{A}$, set of object types $\boldsymbol{T}$ relevant to $\boldsymbol{A}$, and an object-event association matrix τ. We use the symbol # for the cardinality of a set or a bag.

3.2.1 Size measures: Informally, the size of a software artifact is a function of the number of finer-grained elements that are used to define, specify, build or compose it. Many size measures for object-oriented software artifacts have been proposed,[7] but as far as we know size has not been expressed and measured in terms of event participations. There are however good reasons to do so:

- Events trigger operations. The more event types an object type is involved in, the more operations must possibly (but not necessarily) be implemented in the class definition. Hence, the count of event participations provides an early size estimate for classes. Early size estimates are useful (and essential) for project budgeting purposes. They are the basis for effort and cost estimates, and for pricing, outsourcing and scheduling decisions.
- In an event-based implementation, early size estimates are needed for the event classes. The number of different types of object affected by an event provides a hint for the size of the event class.
- Empirical software engineering studies have established or corroborated relationships between class size and quality, mainly in terms of defects found [4, 8, 12]. Early size estimates may be used to pinpoint types that are more likely to have error-prone class definitions. For these types, proactive actions can be taken during system design (e.g. using more than one class to implement a 'big' type).
- The size of software artifacts is often used as a normalisation factor, for instance, to express defect densities [21]. Size measurements based on event participations can be used to normalise other measurements (cf. infra).

Table 2. Size measures based on object-event interaction

MEASUREMENT OBJECT	MEASURE DESCRIPTION	MEASURE DEFINITION
Object type: $P \in \boldsymbol{T}$	Count of Event Participations	$CEP(P) = \#\{e \in \boldsymbol{A} \mid \tau(e,P) \neq ' ' \text{ '/' } '\}$
Event type: $e \in \boldsymbol{A}$	Count of Object types Involved	$COI(e) = \#\{P \in \boldsymbol{T} \mid \tau(e,P) \neq ' ' \text{ '/' } '\}$
Conceptual schema: S	Level of Object-Event Interaction	$LOEI(S) = \sum_{P \in \boldsymbol{T}} CEP(P) = \sum_{e \in \boldsymbol{A}} COI(e)$

[6] The informal definitions of many of the measurement concepts have been inspired by previous work of Briand *et al.* [11] and Abreu *et al.* [14]. More formal definitions can be found in [28]. Some aspects of the underlying, Measurement Theory-based, attribute modeling and measure definition processes have been published in [29].

[7] Example measures include DSC (Design Size in Classes -- the count of classes in a design [2]), WMC (Weighted Method Count -- the count of methods in a class [17]), NA (Number of Attributes -- the count of attributes in a class [1]), and SCI (Size of Class Interface -- the count of messages an instance of a class can receive [13]).

3.2.2 Coupling measures: Coupling can be described as the degree of interdependence between software artifacts (e.g. modules, classes, components, etc.). The main arguments in favour of low coupling are that the stronger the coupling between software artifacts, (i) the more difficult it is to understand individual artifacts, and hence to maintain them; (ii) the larger the extent of (unexpected) change and defect propagation effects across artifacts, and consequently the more testing required to achieve satisfactory reliability levels; (iii) the lower the reusability of individual artifacts. In the context of object-oriented software, the disastrous effects of class coupling on quality (e.g. fault-proneness, reusability, etc.) have been demonstrated in a number of empirical studies [3, 8, 12]. We therefore need to assess, and if needed reduce, the level of coupling in a software system. The earlier this is done the better.

Traditionally, coupling in object-oriented design has been measured based on associations and dependency or uses relationships.[8] However, association-based coupling has not been measured in terms of its effect on dynamic aspects (e.g. inheritance and propagation of event participations). Dependency/uses-based coupling has only been measured in terms of message passing. In conceptual modeling we do not wish to decide yet whether object communication will be based on message passing. In our opinion, it might thus be useful to express coupling in terms of common object-event interactions. Object types are then coupled if their instances participate in the same types of event. Two event types can also be coupled, i.e. when object instances of a same type are involved in their respective event occurrences. Further reasons to express and measure coupling in function of common event participation (or common object involvement) are:

- Additional types of coupling, related to the dynamic behaviour of objects, can be measured. An example is synchronisation-based coupling. In **LIBRARY**, RESERVATION and RENEWABLE_LOAN are not (directly) related through associations (cf. Figure 1). However, their alphabets contain the common event type *fetch*. A RESERVATION object and a RENEWABLE_LOAN object synchronise their lives when they participate in the same *fetch* event (i.e. the *fetch* ends the life of a RESERVATION object and creates a new RENEWABLE_LOAN object).
- Measurements are not restricted to 'direct' coupling relationships [9].

Table 3. Coupling measures based on object-event interaction

MEASUREMENT OBJECT	MEASURE DESCRIPTION	MEASURE DEFINITION
Object type: $P \in \boldsymbol{T}$	Count of Coupled Object types	$CCO(P) = \#\{Q \in \boldsymbol{T} - \{P\} \mid \exists e \in \boldsymbol{A}: \tau(e,P) \neq \text{' '} \wedge \tau(e,Q) \neq \text{' '}\}$
Event type: $e \in \boldsymbol{A}$	Count of Coupled Event types	$CCE(e) = \#\{e' \in \boldsymbol{A} - \{e\} \mid \exists P \in \boldsymbol{T}: \tau(e,P) \neq \text{' '} \wedge \tau(e',P) \neq \text{' '}\}$
Conceptual schema: S	Level of Object type Coupling	$LOC(S) = \sum_{P \in \boldsymbol{T}} CCO(P) / 2$
	Level of Event type Coupling	$LEC(S) = \sum_{e \in \boldsymbol{A}} CCE(e) / 2$

[8] For a comprehensive overview of class coupling measures we refer to [9]. Well-known coupling measures for object-oriented software include CBO (Coupling Between Objects -- the count of classes to which a class is coupled, including inheritance-based coupling [17]), MOA (Measure Of Aggregation -- the count of the number data declarations whose types are user defined classes [2]), MPC (Message Passing Coupling -- the count of send messages defined in a class [23]), and COF (COupling Factor -- a normalised measure of the count of coupled object class pairs [13]).

3.2.3 Inheritance, specialisation and propagation measures: Several measures for quantifying the absolute or relative amount of inherited properties in an object-oriented system have been proposed.[9] Empirical research has shown that the inheritance-related aspects captured by these measures are related to quality aspects (e.g., fault-proneness of a class) [15]. Moreover, inheritance is a form of reuse, and might thus have an effect on development effort.

The measures proposed in the literature are generally design or code measures that consider the inheritance of class methods. During object-oriented analysis, models are built using type definitions, rather than class definitions, and consequently it has not been decided yet which operations must be implemented, or inherited, overridden, etc. Nevertheless, early estimates of the degree of inheritance can be obtained by considering the type of provenance of event participations. Generally, operations that correspond to 'own' or specialised event participations are implemented as class methods, whereas inherited operations are not (unless there is a need to override the method body in the subclass).

As far as we know, there are no measures to assess the degree of propagation of operations. Nevertheless, propagation of operations is not an exclusive characteristic of existence dependency associations. For instance, in the context of the IS-PART-OF relation operations might also propagate from the aggregate to the parts. Having an idea of the (relative) amount of propagated operations in an object type is useful, as these operations must not necessarily be implemented as methods in the own class definition.

Table 4 contains measures for the degree of inheritance, specialisation and propagation of event participations, both for object types and conceptual schemes. For the sake of brevity, similar measures for the event perspective are not presented.

Table 4. Inheritance, specialisation and propagation measures based on object-event interaction

MEASUREMENT OBJECT	MEASURE DESCRIPTION	MEASURE DEFINITION
Object type: $P \in \boldsymbol{T}$	Degree Of Inheritance	$DOI(P) = \#\{e \in \boldsymbol{A} \mid \tau_P(e,P) = I\}/CEP(P)$
	Degree Of Specialisation	$DOS(P) = \#\{e \in \boldsymbol{A} \mid \tau_P(e,P) = S\}/CEP(P)$
	Degree of Propagation	$DOP(P) = \#\{e \in \boldsymbol{A} \mid \tau_P(e,P) = A\}/CEP(P)$
Conceptual schema: S	Degree Of Inheritance	$DOI(S) = \sum_{P \in \boldsymbol{T}} \#\{e \in A \mid \tau_P(e,P) = I\}/LOEI(S)$
	Degree Of Specialisation Degree of Propagation	(analogue definitions for DOS and DOP)

[9] Examples: MIF and AIF (Method/Attribute Inheritance Factor -- the ratio of the count of all inherited methods/attributes in the classes of a system to the count of all available methods/attributes in the classes of the system [13]), NIM, NAM and NOM (Number of Inherited/Added/Overriding Methods [36]), and MFA (Measure of Functional Abstraction -- the ratio of the number of methods inherited by a class to the total number of methods accessible by member methods of the class [2]).

3.2.4 Polymorphism measures: Literally, polymorphism refers to the ability to take different forms. In object orientation the term 'polymorphism' is itself polymorphic as many flavours of polymorphism have been identified. The general idea of polymorphism is that different classes define a method with the same name and/or signature, but with a different implementation. Mostly, polymorphism is then considered in the context of inheritance and overriding, combined with either dynamic binding (in case the signature is the same) or static binding (in case the signature is different). Inheritance-related polymorphism is also the type of polymorphism measured by most of the published measures.[10] However, methods with the same name and/or signature may also appear in classes not related through inheritance.[11]

The effect of polymorphism on quality has been empirically investigated [5, 15]. For instance, the study of Benlarbi *et al.* [5] suggests that, contrary to popular belief, polymorphism may increase the probability of faults in software. It is therefore important to assess and control the degree of polymorphism in a system, preferably in the early phases of system development.

Again, the object-event association matrix contains the necessary information to measure polymorphism during conceptual modeling. A potential polymorphic situation exists when more than one object type is involved in the same event type. So, the number of potential polymorphic situations is easily measured from the perspective of the event types. The type of provenance indications further allow to distinguish between types of polymorphism, like inheritance-related polymorphism and non-inheritance-related polymorphism (also called simultaneous or reciprocal polymorphism).

Table 5 presents the degree of polymorphism measures. They are relative measures, i.e. they relate the actual number of potential polymorphic situations of the type considered to the maximum number of such situations. Since all published polymorphism measures are defined at the system level, we use only conceptual schemes as measurement objects.

Table 5. Polymorphism measures based on object-event interaction

MEASUREMENT OBJECT	MEASURE DESCRIPTION	MEASURE DEFINITION
Conceptual schema: S	Degree of POlymorphism	$DPO(S) = (\sum_{e \in A} COI(e) - \#A)/LOEI(S)$
	Degree of Inheritance-related POlymorphism	$DIPO(S) = \sum_{e \in A} \#\{P \in T \mid \tau_P(e,P) = I\} / \sum_{e \in A} \#\{P \in T \mid \tau_P(e,P) = S \vee \tau_P(e,P) = I\}$
	Degree of Non-inheritance-related POlymorphism	$DNPO(S) = (\sum_{e \in A} \#\{P \in T \mid \tau_P(e,P) \neq I \wedge \tau(e,P) \neq \text{' '/' '}\} - \#A) / (LOEI(S) - \sum_{e \in A} \#\{P \in T \mid \tau_P(e,P) = S \vee \tau_P(e,P) = I\})$

[10] For instance, the POlymorphism Factor (POF) of Abreu *et al.* [13] is described as the ratio of the actual number of possible different polymorph situations to the maximum number of possible distinct polymorph situations, where the notion of a polymorph situation is restricted to cases where a method of a class is overridden in a descendant. The polymorphism measures of Periyasamy *et al.* [26] focus also exclusively on method overriding.

[11] As far as we know, only Benlarbi *et al.* [5] have defined a measure for non-inheritance-related polymorphism.

4. Conclusions and topics for further research

The central piece of the conceptual measurement framework is the object-event association matrix. In fact, to measure characteristics related to the data, function, as well as dynamic behaviour dimensions of software, the framework assumes that the object-event interactions are somehow modeled. Information regarding such interactions is normally available when modeling and analysing a domain or system. But even if a method does not contain an object-event association matrix or an equivalent object-event interaction schema, this information can easily be derived, as long as the method does not ignore behavioural aspects altogether. In our opinion, popular domain analysis, CBSE and OOAD methods like Catalysis and the Rational Unified Process contain dynamic concepts that can be mapped to the concept of an event. Some of the dynamic concepts used (e.g. use cases, scenarios) must first be further decomposed to the level of atomic actions and events. Next, actions must be transformed into events having no duration. Developing methodological and tool support for deriving an object-event interaction schema could be one topic for further research.

Other topics for further research include a variety of empirical investigations. It would be useful to examine whether OOA measures can indeed be used as early effort and quality predictors, and whether dynamic aspects of software are related to these variables. One or our current projects, related to this type of research, investigates the use of the proposed measures for the componentisation of object models.

Acknowledgements

Geert Poels is a Postdoctoral Fellow of the Fund for Scientific Research - Flanders (Belgium)(F.W.O.) and wishes to acknowledge the financial support of the Fund for Scientific Research.

References

[1] L. Badri, M. Badri, and S. Ferdenache, "Towards Quality Control Metrics for Object-Oriented Systems Analysis", *Proceedings of TOOLS Europe'95*, Versailleş, France, March 1995, pp. 193-206.

[2] J. Bansiya and C. Davis, "An Object-Oriented Design Quality Assessment Model", Research Report, Computer Science Department, University of Alabama, Huntsville, AL, USA, August 1997 (accepted for publication in *IEEE Transactions on Software Engineering*).

[3] V.R. Basili, L. Briand, and W.L. Melo, "A Validation of Object-Oriented Design Metrics as Quality Indicators", *IEEE Transactions on Software Engineering*, Vol. 22, No. 10, October 1996, pp. 751-761.

[4] S. Benlarbi, K. El Emam, and N. Goel, "Issues in Validating Object-Oriented Metrics for Early Risk Prediction", *Proceedings of the 10th International Symposium on Software Reliability Engineering (ISSRE'99)*, 1999.

[5] S. Benlarbi and W.L. Melo, "Polymorphism Measures for Early Risk Prediction", *Proceedings of the 21st International Conference on Software Engineering (ICSE'99)*, Los Angeles, May 1999, pp. 334-344.

[6] G. Booch, J. Rumbaugh, and I. Jacobson, *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.

[7] L. Briand, E. Arisholm, S. Counsell, F. Houdek, and P. Thévenod-Fosse, "Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of The Art and Future Directions", Technical Report IESE 037.99/E, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany, 1999 (ICSE'99 workshop on Empirical Studies of Software Development and Evolution (ESSDE), Los Angeles, May 1999).

[8] L.C. Briand, J.W. Daly, V. Porter, and J. Wüst, "A Comprehensive Empirical Validation of Product Measures for Object-Oriented Systems", Technical Report ISERN-98-07, Fraunhofer Institute for Experimental Software Engineering, Kaiserslautern, Germany, 1998.

[9] L.C. Briand. J.W. Daly, and J.K. Wüst, "A Unified Framework for Coupling Measurement in Object-Oriented Systems", *IEEE Transactions on Software Engineering*, Vol. 25, No. 1, Jan-Feb. 1999, pp. 91-121.
[10] L. Briand, P. Devanbu, and W. Melo, "An investigation into Coupling Measures for C++", *Proceedings of the 19th International Conference on Software Engineering (ICSE'97)*, Boston, May 1997.
[11] L. Briand, S. Morasca, and V.R. Basili, "Property-Based Software Engineering Measurement", *IEEE Transactions on Software Engineering*, Vol. 22, No. 1, January 1996, pp. 68-86.
[12] L.C. Briand, J. Wüst, S. Ikonomovski, and H. Lounis, "A Comprehensive Investigation of Quality Factors in Object-Oriented Designs: an Industrial Case Study", *Proceedings of the 21st International Conference on Software Engineering (ICSE'99)*, Los Angeles, May 1999.
[13] F. Brito e Abreu and R. Carapuça, "Object-Oriented Software Engineering: Measuring and Controlling the Development Process", *Proceedings of the 4th International Conference on Software Quality (ICSQ'94)*, McLean, VA, USA, October 1994.
[14] F. Brito e Abreu, R. Esteves, and M. Goulao, "The Design of Eiffel Programs: Quantitative Evaluation Using the MOOD Metrics", *Proceedings of TOOLS'96*, Santa Barbara, Calif., USA, July 1996.
[15] F. Brito e Abreu and W. Melo, "Evaluating the Impact of Object-Oriented Design on Quality", *Proceedings of the 3rd International Software Metrics Symposium (METRICS'96)*, Berlin, March 1996.
[16] F. Brito e Abreu, H. Zuse, H. Sahraoui, and W. Melo, "Quantitative Approaches in OO Software Engineering", ECOOP'99 Workshop Reader, Lecture Notes in Computer Science, Springer Verlag, 1999 (ECOOP'99 workshop, Lisbon, June 1999).
[17] S.R. Chidamber and C.F. Kemerer, "A Metrics Suite for Object Oriented Design", *IEEE Transactions on Software Engineering*, Vol. 20, No. 6, June 1994, pp. 476-493.
[18] D. Coleman *et al.*, *Object-Oriented Development: The Fusion Method*, Prentice-Hall, 1993.
[19] S. Cook and J. Daniels, *Designing Object Systems: Object-Oriented Modeling with Syntropy*, Prentice-Hall, 1994.
[20] D.F. D'Souza and A.C. Wills, *Objects, Components, and Frameworks with UML: the Catalysis Approach*, Addison-Wesley, 1999.
[21] N.E. Fenton and S.L. Pfleeger, *Software Metrics: A Rigorous & Practical Approach*, PWS Publishing Company, London, 1997.
[22] I. Graham, *Migrating to Object Technology*, Addison-Wesley, 1995.
[23] W. Li and S. Henry, "Object-Oriented Metrics that Predict Maintainability", *Journal of Systems and Software*, Vol. 23, No. 2, November 1993, pp. 111-122.
[24] J. Martin and J. Leben, *Strategic Information Planning Methodologies*, Prentice-Hall, 1989.
[25] B. Meyer, *Object-Oriented Software Construction*, 2nd edition, Prentice-Hall, 1997.
[26] K. Periyasamy and X. Liu, "A New Metrics Set for Evaluating Testing Efforts for Object-Oriented Programs", *Proceedings of TOOLS'99*, Santa Barbara, Calif., USA, August 1999, pp. 84-93.
[27] G. Poels, "On the use of a Segmentally Additive Proximity Structure to Measure Object Class Life Cycle Complexity", in: R. Dumke and A. Abran (Eds.), *Software Measurement: Current Trends in Research and Practice*, Deutscher Universitäts Verlag, Wiesbaden, Germany, 1999, pp. 61-79.
[28] G. Poels, "On the Formal Aspects of the Measurement of Object-Oriented Software Specifications", Ph.D. dissertation, Catholic University of Leuven, Belgium, April 1999.
[29] G. Poels and G. Dedene, "Distance-based software measurement: necessary and sufficient properties for software measures", *Information and Software Technology*, Vol. 42, No. 1, January 2000, pp. 35-46.
[30] Rational Software, Object Oriented Analysis and Design, Student Manual, 1998, http://www.rational.com/.
[31] K. Robinson and G. Berrisford, *Object-oriented SSADM*, Prentice-Hall, 1994.
[32] J. Scholtz *et al.*, "Object-Oriented Programming: The Promise and the Reality", *Journal of Systems and Software*, Vol. 23, No. 2, November 1993, pp. 199-204.
[33] A.J.H. Simons, M. Snoeck, and K.S.Y. Hung, "Design Patterns as Litmus Paper to Test the Strength of Object-Oriented Methods", *Proceedings of the 1998 International Conference on Object Oriented Information Systems (OOIS'98)*, Paris, September 1998.
[34] M. Snoeck and G. Dedene, "Existence Dependency: The Key to Semantic Integrity Between Structural and Behavioural Aspects of Object Types", *IEEE Transactions on Software Engineering*, Vol. 24, No. 4, April 1998, pp. 233-251.
[35] M. Snoeck, G. Dedene, M. Verhelst, and A. Depuydt, *Object Oriented Enterprise Modelling with MERODE*, Academic Press Leuven, Belgium, 1999.
[36] D.P. Tegarden, S.D. Sheetz, and D.E. Monarchi, "A Software Complexity Model of Object-Oriented Systems", *Decision Support Systems: An International Journal*, Vol. 13, 1995, pp. 241-262.
[37] H. Zuse, *A Framework for Software Measurement*, Walter de Gruyter, Berlin, 1998.